REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
02-07-2005	Final Report	Jan 2004 - May 2005

4. TITLE AND SUBTITLE
Applying Light Mapping Techniques to Vis-Sim Databases

5a. CONTRACT NUMBER
W56HZV-04-C-0101

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Mr. Michael M. Morrison

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
RealTime Technologies, Inc.
1517 N. Main St.
Royal Oak, MI 48067

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army TARDEC
National Automotive Center
6501 E. 11 Mile Road
AMSRD-TAR-N MS: 157
Warren, MI 48397-5000

10. SPONSOR/MONITOR'S ACRONYM(S)
TARDEC - NAC

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
#14904

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unlimited Distribution

13. SUPPLEMENTARY NOTES
Unclassified

14. ABSTRACT

The Gaming and Visualization and Simulation industries have recently focused on shaders and stencil shadow volumes to add realistic lighting and hard shadow effects to the scene. While producing great looking results these techniques are still computationally expensive at run time, especially when simulating many light sources. For most databases and simulations the lighting and shadow effects for the terrain geometry are diffuse, static in nature, and not greatly affected by dynamic entities. Using this assumption the diffuse lighting, shading, and shadowing effects of all static light sources can be pre-rendered into the database. This technique is generally known as "Light Mapping" and has been in wide use in the gaming industry since ID Software's Quake 1. This paper will present the tool set and methods that RTI developed in order to apply this technique to OpenFlight models. This work was supported by an SBIR Phase I Option contract (No. W56HZV-04-C-0101) in conjunction with the US Army TARDEC.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Alexander A. Reid
Unclassified			N/A	14	19b. TELEPHONE NUMBER *(Include area code)* 586-753-2212

14904

APPLYING LIGHT MAPPING TECHNIQUES TO VIS-SIM DATABASES

Michael M. Morrison
Sr. Research Scientist
Realtime Technologies, Inc.

Abstract

The Gaming and Visualization and Simulation industries have recently focused on shaders and stencil shadow volumes to add realistic lighting and hard shadow effects to the scene. While producing great looking results these techniques are still computationally expensive at run time, especially when simulating many light sources. For most databases and simulations the lighting and shadow effects for the terrain geometry are diffuse, static in nature, and not greatly affected by dynamic entities. Using this assumption the diffuse lighting, shading, and shadowing effects of all static light sources can be pre-rendered into the database. This technique is generally known as "Light Mapping" and has been in wide use in the gaming industry since ID Software's Quake 1. This paper will present the tool set and methods that RTI developed in order to apply this technique to OpenFlight models. This work was supported by an SBIR Phase I Option contract (No. W56HZV-04-C-0101) in conjunction with the US Army TARDEC.

Introduction

A "Light Map" is simply a texture map that encompasses the diffuse lighting effects in an environment. Light Maps can be intensity maps as shown below, or can be full RGB images for use with colored light sources.

Figure 1 - Light Mapping [11]

Figure 1 depicts the simple multi-texture operation used in Light Mapping. The image on the left is the decal texture, the image in the middle is the Light Map, and the image on the right is the product of the two. The intensity of the Light Map modulates the attenuation of the surface providing the diffuse lighting effect - in the same way vertex colors do in the OpenGL lighting equation.

The series of screen shots in Figure 2 show the texture compositing from the decal to the Light Map to the final result in a driving simulation database.

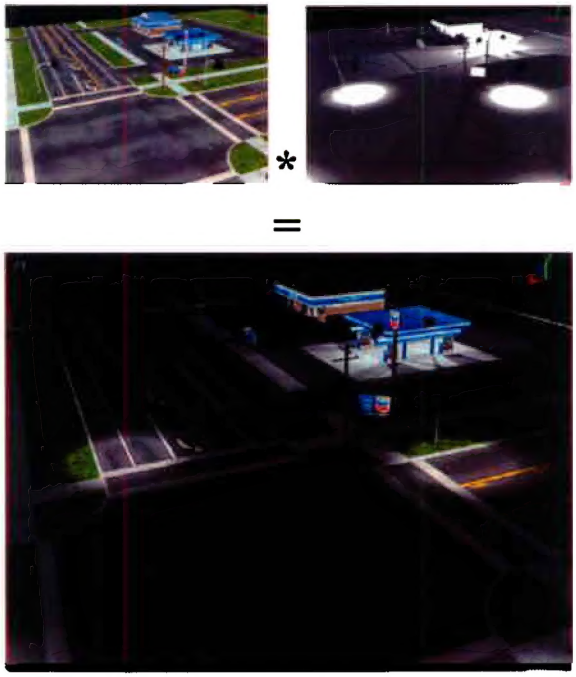

Figure 2 - Light Mapped OpenFlight Database [GB]

Note the soft shadows cast on the terrain surface around the gas pumps and the Phong-shaded spot light effects from the street lights. Four spot lights and two point light sources were used to render this database.

While this technique has been around for some time, recently released games still use Light Mapping because it is so compelling.

Figure 3 – Counter Strike [12]

Figure 3 shows an outdoor scene from the popular PC video game "Counter Strike - Source" by Valve Software. Counter Strike - Source is based on the Half Life 2 engine which uses Light Mapping extensively for

shadowing and local lighting effects. Half Life 2 was released in late 2004 / early 2005.

Light Mapping Construction

Light Maps are constructed in a pre-processing step potentially using detailed per-pixel lighting and "soft" shadowing models. An individual Light Map texture is rendered for every polygon in the database using either a "local" (also called 'direct') or "global" illumination model. The resolution of these light maps is based on the lumel per meter ratio assigned to the particular surface, as described later. The individual Light Maps are then packed into some minimal number of larger texture maps to enhance runtime performance.

Only diffuse lighting effects can be pre-computed because they are view-independent. This means that Light Mapping is generally not useful to pre-render environment-mapped effects and specular lighting as they are both view-position-dependent. Methods to add several view-dependent effects are presented in a later section.

Illumination Model

Light Maps are generally sampled over a regular grid, similar to that shown in Figure 4.

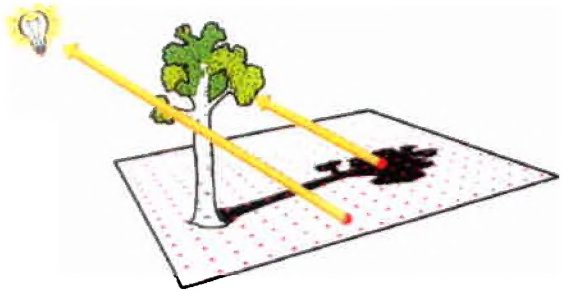

Figure 4 - Lumel Sampling

Light Map samples are known as "lumels" and are shown as red dots above. A lumel represents one texel in the final Light Map texture. Rays are cast from each lumel to every light source in the scene, and an illumination model is run to determine the lighting intensity and color at that point. The illumination model may take shadowing into account as well as other environmental effects such as clouds and sky color.

A lumel per meter ratio is pre-selected for each surface based on the desired resolution of the shadow or light source effects on that surface. A greater lumel per meter ratio will provide greater detail in the final Light Map, but will require more texture memory and more rendering time.

Local Illumination

OpenGL's fixed-function pipeline uses the equation shown in Figure 5 to compute per-vertex lighting.

$$
\begin{aligned}
vertex\_colcr = {} & \\
& emission_{material} + \\
& ambient_{lightmodel} * ambient_{material} + \\
& \sum_{i=0}^{n-1} \left(\frac{1}{k_c d + k_l d + k_q d^2}_i * \right. \\
& \left. \left(\begin{array}{c} ambient_{material} + \\ \max(L_i \bullet n, 0) * diffuse_{light_i} * diffuse_{material} + \\ \max(s_i \bullet n, 0)^{shiness} * specular_{light_i} * specular_{material} \end{array} \right)_i \right)
\end{aligned}
$$

Figure 5 - OpenGL Lighting Equation [cite]

The Local Illumination model computes lighting per-pixel using an equation similar to that of Figure 5, minus the specular term. Most Light Mapping applications also simplify the material and emissive terms as well, assuming they are either 1.0 or 0.0 as appropriate. In addition to the luminance, the model also takes into account shadowing effects. Pixels in-shadow from a particular light source are given a default "shadow ambient" value that represents the indirect lighting contribution at that point.

It is also possible to mark certain surfaces as having a full intensity light map to appear uniformly lit, rather than being affected by any light sources in the scene. This is similar to the effect in Multigen Creator of setting a face's Shade Mode to "Flat."

Figure 6 is a portion of the Light Map from the scene in Figure 2. It was computed using the Local Illumination model.

Figure 6 - Local Illumination

The white rectangle in the center is the illuminated Chevron station sign. It has been marked as having a full intensity Light Map, but does not have a light source associated with it. As shown in Figure 2 this makes the Chevron station sign appear fully lit. The

illuminated area to the right is from the street light, which is set up as a spot light source.

Global Illumination

Global Illumination takes the indirect lighting of all surfaces into account when computing the Light Map color and intensity. One of the more popular global illumination models is known as "Radiosity". In the book "Real-Time Rendering" Thomas Moller describes Radiosity: "Light bounces around in an environment; you turn a light on and the illumination quickly reaches a stable state. In this stable state, each surface can be considered as a light source in its own right. When light hits a surface, it can be absorbed, diffusely reflected, or reflected in some other fashion (specularly, anisotropically, etc). Basic radiosity algorithms first make the simplifying assumption that all indirect light is from diffuse surfaces. This assumption fails for places with polished marble floors or large mirrors on the walls, but for most architectural settings, this is a reasonable approximation."[CITE: R-TR, p278]

Some packages render radiosity directly, others calculate radiosity using the results of a local illumination model. Figure 7 is a radiosity rendering using the results of Figure 6 as a baseline.

Figure 7 - Global Illumination

This Light Map shows illumination in the area in front of the Chevron sign whereas Figure 6 does not.

Figure 8 – LI (above) VS GI (below)

The images in Figure 8 show the differences in the scene when applying the decal texture as well. The image below looks more realistic due to the reflected lighting contribution from the Chevron sign.

Global Illumination takes significantly longer to render than Local Illumination (possibly 10x or better) due to the additional computation involved. However, there is no additional runtime penalty.

Light Map Packing

Once individual surface Light Maps have been rendered they must be packed into a larger texture map.

Figure 9 shows a Light Mapped house – a free OpenFlight model downloaded from CGSD's web site.

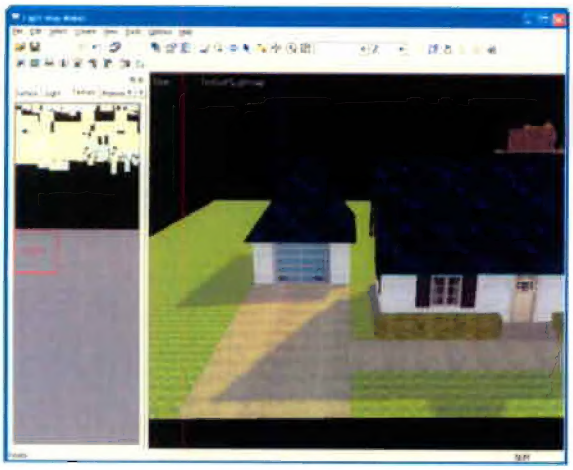

Figure 9 - Light Mapped House

The window on the left hand side of the figure shows the single packed Light Map for the scene. This scene was rendered using a single directional light source that was yellow in color, producing the yellow and gray light map.

Final Light Map resolution is determined by the lumel per meter ratio and the number of polygons in the database. The number of light maps will further be determined by the maximum texture resolution allowed on the particular architecture. In the PC world, Nvidia allows a maximum resolution of 4096x4096, while ATI's maximum resolution is 2048x2048.

Justification

Light Mapping Advantages

The Light Mapping technique has several advantages over per-vertex lighting, shader-based lighting, and either stencil shadow volumes or shadow mapping techniques, especially for terrain lighting.

Unlike dynamic effects such as shaders and shadow volumes or shadow mapping, Light Maps are generally pre-computed and static. This generally means a fixed "time of day" is pre-computed as well as fixed shadow angles and fixed light source state and position.

While long simulation runs where time of day must change appropriately is an issue for some, fixed time of day (and therefore shadow angles and light states) has not generally been a problem for Realtime Technologies. Several dynamic effects, including the ability to change a light source's state, will be addressed later.

As a side note, "night time" scenes have been selected to illustrate Light Mapping in most images because they show off the technique very well. Light Mapping is equally effective in daylight scenes, as shown in Figure 9 and Figure 3.

The following sections will compare the advantages and disadvantages of Light Mapping verses other techniques.

Vertex Lighting

Vertex lighting can produce scenes that are visually compelling. The quality of vertex lighting approaches is generally dependent on the types of light sources required and the tessellation level of the database. Infinite directional lights, such as the Sun, tend to work reasonably well with vertex lighting, whereas spot light sources only work well if a reasonable collection of terrain vertices lie in the vicinity of the spot light cone. Point lights can have the same problems as spot lights depending on the desired effect. In addition, the vertex lighting model of OpenGL, and most image generators, does not take occlusion and/or shadowing into account.

Multigen-Paradigm's Creator provides a feature allowing the user to "Calculate Shading." This can be used to update vertex colors to "bake" the lighting into the model. Figure 10 shows the Chevron station lit using this approach.

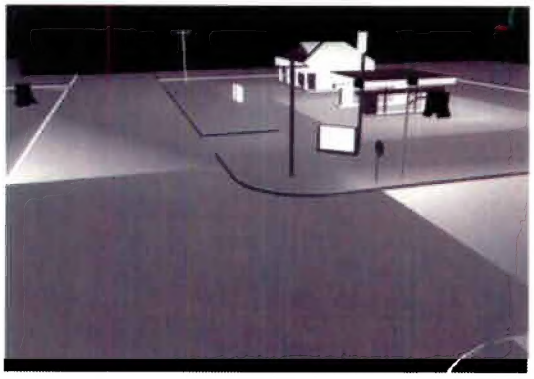

Figure 10 - Creator Vertex Lighting

This scene shows the spot light sources as having little or no effect on the intersection polygon, as none of the verts lie directly within the spot light cone. This is one of the typical issues presented by vertex lighting. There is also no scene shadowing. However, this model will run on any Image Generator that understands OpenFlight files at full speed.

Figure 11 shows the Light Maps rendered from the same set of light sources used in the previous image.

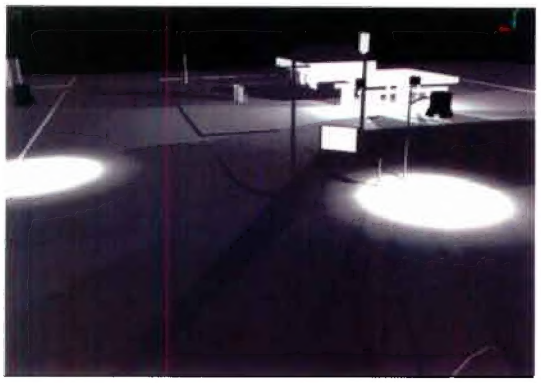

Figure 11 – Light Mapping

The outlines of the spot light cones from the street lights are clearly visible, as are the shadows cast from various elements in the scene.

Tests at Realtime Technologies have shown that issuing a secondary texture map, with or without vertex colors, produces a performance penalty of less than 10% on older ATI Radeon 9800 class hardware. The performance difference may be less on newer graphics cards.

Shader-Based Lighting

Shaders are extremely useful. They are definitely "what's next" and work well for a fair number of situations right now. RTI uses shaders in a number of applications, even in conjunction with Light Mapping. In the future shaders will definitely replace Light Mapping as the rendering method of choice. However, the runtime efficiency of a multi-texture operation cannot currently be matched by shaders for the situations where Light Mapping works well. This translates to more effective light sources and greater geometric detail in a given scene.

By way of comparison, the following two (conceptual) GLSL shaders show the code necessary to implement light mapping and the code necessary to implement a single real-time Phong-shaded diffuse light source.

Vertex Shader

```
void
main ( void )
{
 # Pass along textures and color
 gl_TexCoord[0] = gl_MultiTexCoord0;
 gl_TexCoord[1] = gl_MultiTexCoord1;

 # Transform Position
 gl_Position = ftransform();
}
```

Fragment Shader

```
void
main( void )
{
 # sample textures
 decalC = tex2D(decalMap, gl_TexCoord[0].st);
 lightC = tex2D(lightMap, gl_TexCoord[1].st);

 # output color value
 gl_FragColor = decalC * lightC;
}
```

Figure 12 - Light Mapping Shader

The vertex shader in Figure 12 simply passes through the values of the texture coordinates for the decal and light map textures, and transforms the gl_Position using the builtin ftransform() method. The fragment shader samples the two textures and sets the fragment color to be their product. Note that no per-vertex normals are necessary, nor are any material colors for lights or polygonal surfaces.

In cases where an intensity map is used as the decal texture an additional color may need to be passed per vertex and multiplied in the fragment shader.

Vertex Shader

```
void
main( void )
{
 # pass along decal texture and color
 gl_TexCoord[0] = gl_MultiTexCoord0;

 # Transform Position
 gl_Position = ftransform();

 # pass along eye space vert pos & norm
 N  = gl_NormalMatrix * gl_Normal;
 EP = gl_ModelViewMatrix * gl_Vertex;
 LP = gl_ModelViewMatrix *
      gl_LightSource[0].position;
}
```

Fragment Shader

```
void
main( void )
{
 # sample texture
 decalC = tex2D(decalMap, gl_TexCoord[0].st);

 # interpolated over the surface
 L = LP - EP;
 N = normalize(N);
 D = 1 / length(L); // hack attenuation
 L = normalize(L);

 diffuse = clamp(dot(L, N), 0.0, 1.0) * D;

 gl_FragColor =
      decalC *
     (( gl_FrontMaterial.ambient *
       gl_LightSource[0].ambient ) +
     (gl_FrontMaterial.diffuse +
      gl_LightSource[0].diffuse ) *
      diffuse);
}
```

Figure 13 - Single Point Light Shader

The vertex shader in Figure 13 is significantly more complex. First, it passes on the texture coordinates of

the decal texture. It must then transform the incoming normal into eye coordinates, the incoming vertex into eye coordinates, and the light position into eye coordinates. These values are linearly interpolated across the surface for the fragment shader.

The fragment shader first samples the decal texture. It must then compute the vector from the surface to the light, and normalize the interpolated surface normal. 'D' is a hack linear attenuation factor for the point light. After N and L are normalized their dot product determines the lighting contribution, with D attenuating it based on distance. The final fragment color is computed using the light source and polygonal ambient and diffuse materials as well as the light source diffuse and ambient factors. Finally, this value is multiplied by the decal texture sample to get the result. The contribution due to specular effects has deliberately been eliminated from this example, as specular lighting cannot be rendered into a Light Map. Note that per-vertex normals are required, and per-polygon materials may be required depending on the desired effect.

As is evident from the conceptual shaders the light mapping approach is extremely efficient compared to just a single Phong-shaded diffuse point-light source. Especially considering that any number of light sources can be represented within a single light map texture, and the light source shader does not compute any shadows.

The advantage to the shader code, however, is that it will light any element in the scene, including dynamic entities. Scene optimizations are possible with this technique, such as virtualized OpenGL light sources and/or other techniques to reduce the computational burdon. With few light sources in the scene this technique works well, but it does not scale as well as Light Mapping.

Dynamic Shadowing

Dynamic shadowing techniques such as stencil shadow volumes and shadow mapping can be efficient and produce good looking results. A direct comparison of the techniques used to compute the shadows themselves will only be briefly discussed here.

The shadow mapping technique is inherently a two pass algorithm, requiring rendering the scene once from the lights perspective and again from the view perspective for all geometry to cast or receive shadows. After rendering from the lights perspective, the Z-Buffer contents are stored and used in a fragment comparison operation while rendering from the view perspective. This comparison determines whether the fragment is in-shadow. It can be tricky to render differing light source types and multiple light sources using this technique.

The stencil shadow mapping technique requires that silhouette edges of all objects casting shadows in the scene be traced and intersected with some distant clipping plane. Geometric faces are then built from the silhouette edges to the intersected clipping planes. This geometry is then rendered into the stencil buffer using various methods (ZPass, ZFail) and various operations for the front and back faces of this geometry. Finally, a single full-screen translucent polygon is rendered, with the stencil buffer enabled, producing a "shadowed" effect everywhere it was drawn to the scene. There are certainly variations on this technique, but this is the basic idea.

Both of these techniques work well to allow dynamic geometry to shadow the terrain. They are somewhat wasteful for static geometry, especially in the cases of fixed light sources and immobile terrain and buildings.

While it will not be covered here, using a shader it is possible to combine these shadowing techniques with light mapping to produce shadowed effects from dynamic entities and mix them with the pre-rendered light maps. This is the technique that Half Life 2 uses to shadow dynamic entities.

Light Map Rendering Tools

As mentioned earlier, this work was funded by an SBIR Phase I Option. Due to budgetary constraints it was not practical for RTI to write its own Light Map rendering tool. The ultimate goal was the ability to Light Map "industry standard" OpenFlight format databases, with as little modification to them as possible. OpenFlight version 15.7 introduced multi-texturing, so it was clear that it would be a viable output format.

An initial search found no commercial or open source Light Mapping or Radiosity rendering packages that supported OpenFlight directly. The majority of the packages found only supported simple file formats such as OBJ or 3DS. Desiring to support LODs and other OpenFlight hierarchy it would have been difficult to convert files to and from these formats for pre-processing. RTI decided that it was necessary to find a tool that would allow a custom database importer and exporter that could be used with OpenFlight.

Many light mapping tools were identified, and several were evaluated. Ultimately, Light Map Maker (LMM) by Windssoft was selected as the tool of choice for the Phase I Option work. Gile[s] by Mikkel Fredborg came in a close second due to its well thought out user interface, and its ability to import multi-textured models.

Review

An internet and industry-wide review of light mapping tools and software was performed in order to select a light mapping product. RTI was specifically looking for a package that would satisfy the following criteria:

1) The tool should be graphical to allow viewing of the rendered database without having to save the file and reload it in a different tool.
2) The tool needs to allow a "plugin" to be written to allow import of OpenFlight files.
3) The tool needs to allow per-face attributes to be stored that will allow the exporter to correlate OpenFlight faces with tool faces.
4) At the minimum, the tool needs to support standard LightMapping techniques.
5) The tool needs to support packing lightmaps in one large texture to minimize OpenGL state change during rendering.
6) Colored light sources should be supported.
7) Soft Shadow casting should be supported.
8) The tool needs to support some method of export "plugin" or write out an open file format that we can use to determine how the light maps were applied to the faces.
9) Preferably, the tool should allow Radiosity rendering in addition to LightMapping.
10) There should be an open relationship with the vendor of the product such that extensions to the product could be negotiated and forthcoming.
11) An Open Source solution is preferred, as this allows RTI to assume control of extensions, if necessary.

A report was prepared summarizing these findings. Both Standalone GUI and Library-based tools were examined and evaluated. Excerpts of this report are presented below.

Stand-Alone Light Map Packages

The programs listed below provide their own graphical user interface, light map rendering, and import/export features. The major function of these packages is to create and export light maps.

Light Map Maker (LMM)

This product stands out immediately for providing a C++ import/export SDK. The import feature allows ancillary data to be passed though facilitating integration with OpenFlight. The standard lighting features include: brightness rectification, blur, expose and smoothing.

LMM supports Radiosity, Mirror, Caustics and standard ray-cast Light Mapping, although the benefits of Mirror and Caustics are questionable. Through the import/export API all scene elements including light sources, geometry, textures, and vertex colors may be imported and exported.

LMM's user interface takes a while to get used to. Some mouse movements seem counter-intuitive, although this is somewhat expected as all 3D interfaces are implemented differently.

Global Illumination Editor (gile[s])

The gile[s] application has a long feature list with a nice GUI. Some of the light map features include: transparency, masking, double sided materials, and a fast sky light (to simulate outdoor scenery). The import plug-in SDK is operational, however no ancillary data is allowed on a per-face basis, and all examples are written in "Blitz Basic". After implementing a partial loader it was discovered that gile[s] was very slow on handling large databases and was therefore abandoned.

Nitrogen LightMapper

Nitrogen is an open source light mapping application. The entire program is essentially a single dialog window with a settings toolbar at the bottom. It is fairly intuitive to use.

The source code for Nitrogen was written in Pascal, and the rendering engine produces visible errors on most scenes. It was determined that using this package would not be a viable option.

Open-Source Radiosity Renderers:

Several open source Radiosity rendering packages were identified.

FSRad

This is an open source radiosity rendering library that comes with several example file import implementations. Out-of-the-box it supports ENT, OCT and ASE file formats, none of which appear to be mainstream. It did not provide a GUI, and does not appear to be actively supported. An amount of work would be required to shape this for use as a user-friendly rendering system, but it is definitely an interesting package especially as a back-end to an existing tool. Commercial usage requires author approval.

RadiosGL

This package is a radiosity rendering library developed primarly under Unix. It supports MDL format files for input, and RGL for output, neither of which is popular or standard. RTI could not get the examples to work properly, nor does it appear this package is actively supported.

Light Map Tools Conclusions

Light Map Maker by Windssoft was chosen as the basis for further development for the following reasons:

1) LMM has a decent GUI; allows plugins; supports soft shadows, colored light sources and light map packing.
2) LMM produces reasonable Light Maps relatively quickly.
3) The focus of this effort is on light maps rather than Radiosity for the time being.
4) Example loaders were included and written in C++.
5) An import loader was already partially written, and as such the Import API was understood.
6) The performance of the GUI was reasonable even when relatively large terrains were loaded.
7) RTI has had contact with the vendor of this product in the past, and they seem fairly responsive.

The following sections will describe how LMM was extended to support the OpenFlight file format, and OpenFlight constructs.

Light Map Maker and OpenFlight

Geometry Import / Export

In order to use LMM with OpenFlight models, custom Importer and Exporter plugins were necessary. Typically, RTI uses a modified version of an Open Source OpenFlight loader for the majority of its work. However, this library is a loader only so an alternate OpenFlight file reader/writer library was required. Multigen-Paradigm's OpenFlight API was selected to provide these capabilities. Using this API it is fairly straightforward to load an OpenFlight file, walk the hierarchy, modify it, and save it back out to disk.

LMM was designed to understand "polygon soup" models – or freeform collections of polygons obeying no particular spatialization or ordering. It does not support hierarchy such as groups, switches or LODs. As such, the importer is not able to specify these things and have them stored along with the geometry. It is therefore not possible to simply take the data structures provided by LMM during an export operation and re-construct the same OpenFlight file that was imported, as the hierarchical information has been lost.

RTI's solution to this problem was the following:

1) Load the OpenFlight file using the OpenFlight API.
2) Scan the model for Faces that satisfy current constraints, mainly the LOD level.
3) Store a pointer to the OpenFlight Face record in the userData field of the polygon being imported into LMM.
4) Retain the OpenFlight model in memory while LMM is modifying it.

Then, during export:

1) Extract the OpenFlight face record pointer from the userData field of each polygon being exported.
2) Extract the secondary texture coordinates from each export polygon vertex and apply them to the correct OpenFlight face vertex.
3) Add the light map texture(s) to the OpenFlight texture palette.
4) Write the database to disk.

The combination of these techniques allows LMM to modify the texture maps and the exporter to write the entire OpenFlight file back to disk, even though LMM does not understand anything about the rest of the OpenFlight hierarchy.

LOD Solution

LOD Import

One of the major differences between OpenFlight and most other 3D file formats is the presence of a structured hierarchy consisting of LODs. LMM's original interface was designed around having a single scene loaded at a time. These scenes contain a collection of geometry, geometric state, and light sources.

The original version of LMM did not support LODs, so RTI worked with Windssoft to introduce the notion of progressive LODs into the LMM product. For each imported surface, the LMM Import API now allows the user to specify an integer LOD level. Using the OpenFlight API it is fairly easy to begin the mgWalk() at the highest detail with mgMostDetail() and continue to add lower LOD levels using mgLessDetail(). Once inside LMM, the GUI will allow viewing of the individual LODs, as well as successively rendering each LOD using the single set of light sources available.

LOD Rendering

Light maps and shadows should be rendered from the highest possible LOD, because this will provide the most resolute imagery. Given that a model broken down into individual LODs as described above will share some polygonal surfaces between LODs, the loader will flag all surfaces with the highest LOD they are a member of. The high LOD will contain many

polygons that are also in lower LODs. Lower LODs will contain fewer and fewer polygons as they will have been previously contained in higher LOD levels. During the import process surfaces that are present in higher LODs are flagged as having "No Light Map," which is an LMM surface option described later. These surfaces are considered during shadow computation, but are not considered for rendering themselves. This also accelerates the rendering process for lower LOD levels. Figure 14 shows the Highest and Lowest LODs from a database.

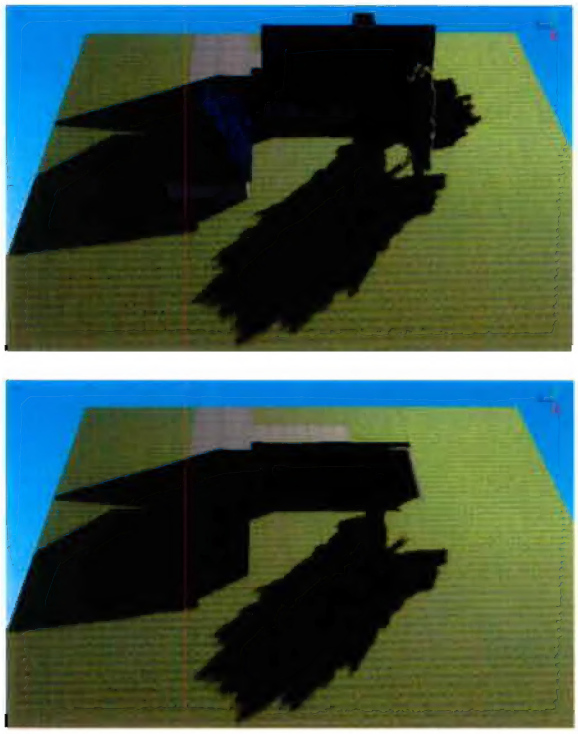

Figure 14 - LOD Rendering [cite]

The upper image is the highest LOD and the lower image is the lowest LOD. The "ground" polygon appears in all the LODs, but has been shadowed only from the highest LOD geometry.

OpenFlight Import Dialog

Several options are presented to the user during an OpenFlight file import. Figure 15 shows the import dialog.

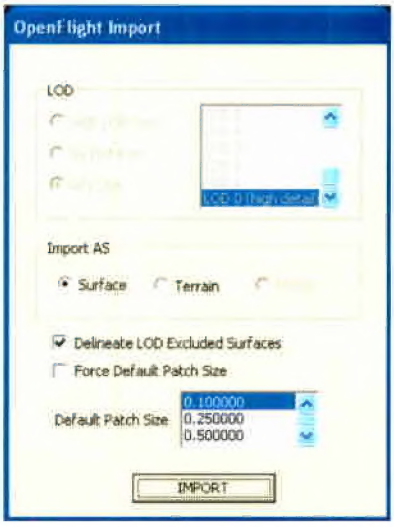

Figure 15 - Import Dialog

The model is scanned and individual LOD levels are presented as options for import. The default import option is to load all LODs. However, it is possible to load, render, and save a single LOD level if desired.

LMM allows several ways to import geometry: Surface, Terrain, or Mobile. "Surfaces" must contain primary decal textures and will have secondary light map textures rendered and applied to them. "Terrains" are rendered using per-vertex lighting as opposed to light maps. This can be useful for large open areas where there will be no shadows cast on or around them. A "Mobile" is an object that does not cast or receive shadows. It is mainly used for LMMs caustics effects. It is possible to simply render a scene to vertex colors by importing a model as a "Terrain." In LMM, as opposed to Creator, shadowing is taken into account when rendering to vertex colors.

The "Patch Size" is the (inverted) meter per texel ratio as described earlier. The dialog allows selection of Patch Size for the entire model, but it can be adjusted on a surface-by-surface basis in the LMM GUI.

Light Sources

The importer and exporter understand OpenFlight and LMM light sources and palettes. Light sources can be added in Creator and imported into LMM, and they can be exported by LMM and read by Creator. Most light source properties have a 1:1 correspondence between the two programs. The light source attributes are converted and imported or exported as described in Table 1.

Light Map Maker Light Source Attribute	OpenFlight Light Source Attribute
Ambient Color – float[4]	Ambient Color – float[4]

Diffuse Color – float[4]	Diffuse Color – float[4]
Specular Color – float[4]	Specular Color – float[4]
Position – float[3]	Position – float[3]
Direction – float[3]	Pitch, Yaw – 2 floats
Attenuation Factors – float[3]	Attenuation Factors – float[3]
Spot Falloff – float	Spot Exponent – float
Type (Point, Spot, Directional)	Type (Local, Spot, Infinite)
Disabled	Disabled
LMM Does not have a notion of this.	Export to Realtime: *Always False*
Range	*OpenFlight does not have the notion of a Range value. Always set to 1000 on import, not exported.*
Inner, Outer Cone	*OpenFlight does not have the notion of an inner and outer spot light cone, therefore both are set to OpenFlight Spot Spread value. On export, Spot Spread is set to the wider of the two values.*

Table 1 - Light Source Conversions

Custom Surface Flags

Light Map Maker supports several per-surface flags that control the light map rendering process:

- *White Lightmap* – Set the color of the light map for this surface to be entirely white. This will cause the surface to appear illuminated, regardless of whether any light source is affecting it. This effect is used in several of the previously rendered scenes for signs and awnings that are supposed to be completely illuminated.
- *No Block Ray* – This surface does not block the light source ray. In other words, the surface can receive shadows and lighting, but does not cast shadows.
- *No Lightmap* – This surface does not receive shadows or light maps, but can cast shadows. This is flag is used in the importer when the surface has already been rendered in a higher LOD level.
- *Patch Size* – The patch resolution in meters per texel.
- *Double Side* – This has the same meaning as the "two-sided" flag in OpenFlight.

These surface attributes are loaded and saved appropriately through the OpenFlight importer and exporter. Since OpenFlight does not contain surface flags for most of the attributes above, the exporter will save these flags to comment fields that are attached to the individual faces in the OpenFlight file. The modeler may modify these within Creator if desired using the format described below.

Fields begin with "#RTI LM", and are followed by the desired settings. Each of the attributes contains an identifier with a value.

- `useWhiteLightmap = {0 or 1}`
  `1 = white light map`
- `noLightmap = {0 or 1}`
  `1 = no light map`
- `noBlockRay = {0 or 1}`
  `1 = do not block ray`
- `patchSize = {floating point number} =`
  `meter per texel ratio`

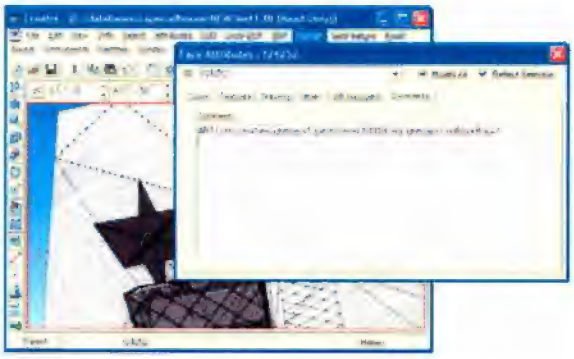

Figure 16 - LMM Comment Fields

The comment field parser will preserve any existing comments contained in the face record, leaving them as the "first" comment in the sequence. This helps older software with broken comment field parsers to ignore RTI's comment field additions.

If nothing is specified in the comment field, the defaults are:

- `useWhiteLightmap = 0`
  `(nc white light map)`
- `noLightmap = 0`
  `(ccmpute light map for surface)`
- `noBlockRay = 0`
  `(block ray on this surface)`
- `patchSize = Value from import dialog.`

The following OpenFlight face flags also control import options:

- `Hidden - Causes surface to be ignored on import.`
- `Twc Sided - Sets the Double Side flag in LMM.`
- `Billboard Flags - Controls Alpha Transparency for the surface. Actual billboards are currently not supported by LMM.`

LMM Rendering Options

Light Map Maker provides a number of different options when rendering Light Maps. The rendering dialog is shown in Figure 17.

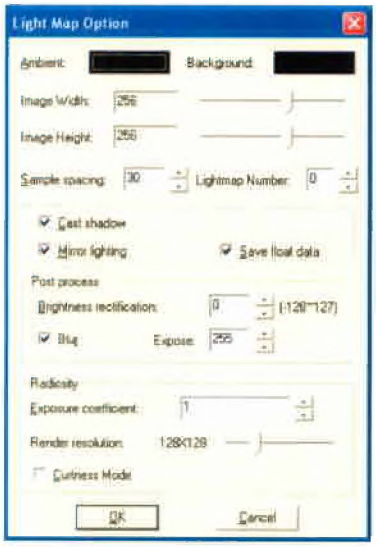

Figure 17 - LMM Rendering Options

Several Important options are described below:

- Ambient: The ambient color used in lighting calculations. This value is applied to the entire scene.
- Image Width, Image Height: The lightmap image resolution. Each Light Map generated will be of this size.
- Sample Spacing: This is an alternate way to specify a global Patch Size for surfaces.
- Lightmap Number: Allows the user to select the desired number of Light Maps. If the value is zero, the number of Light Maps will be determined by the per-surface sample spacing and Image Width and Height. If the value is greater than zero, LMM will uniformly distribute the patch size across all polygons in the scene to create that number of Light Maps.
- Curtness Mode: When rendering with Radiosity, Curtness Mode will run an algorithm that is six times faster than a "full" Radiosity solution. It is much faster, but produces a result that is not as precise.

Scene Graph Support

RTI has tested several Scene Graphs for support of multi-textured OpenFlight files. At this time, Scene Graph Library (SGL) version 0.7.0 and Open Scene Graph (OSG) version 0.9.8-2 are known to correctly load and display multi-textured files.

Multigen-Paradigm Creator loads, displays and manipulates the files correctly, but the "Vega Prime" viewer that ships with Creator 3.0 does not appear to work at all, so it is not clear whether Vega will render the databases correctly.

Performer 3.1 for Windows (OpenFlight loader 15.7 2/1/2003) does not display the databases correctly. It doesn't even attempt to load the texture files, which probably means that it does not understand OpenFlight multi-texture records. SGI-based systems running recent releases of Performer were not available, and therefore not tested.

Database Special Effects

Dynamic Effects

Light Mapping is typically considered to be a "static" effect. Light source states, angles and colors, as well as database geometry, are pre-configured and Light Maps are rendered. Secondary texture coordinates are generated and this information is saved to disk.

However, there are several techniques in use that will allow dynamic modification to Light Mapped environments. The following methods will likely require a shader or custom support from the Scene Graph API, as they are not readily encoded into the OpenFlight file format without significant geometry duplication.

Light Source State

There are several methods that can be used to modify a light source's state at runtime, two of which are texture swapping and direct texture modification.

The direct texture modification method requires knowledge of the mapping between lumels in the Light Map, polygons in the scene, and light sources in the scene. With this information, Light Map lumels could be updated dynamically based on light state changes. This method provides a great deal of flexibility, as new light sources could even be added at runtime. While this method has merit, it is more complex than the texture swapping method and requires the most support from the application and Scene Graph API. Several games including Quake and Half Life make use of this technique to modify light maps on the fly.

If there are relatively few light states that need to be encoded, the texture swapping method can be used to provide dynamic lighting effects. Assuming a consistent texture coordinate mapping, the texture swapping method replaces an entire Light Map representing one state with another Light Map representing a second state – similar to the way other animated texture mapping tricks operate.

This technique can be particularly effective if the environments are "tile based" and light sources from one tile would not affect another. This allows each tile to be rendered individually with varying light state permutations.

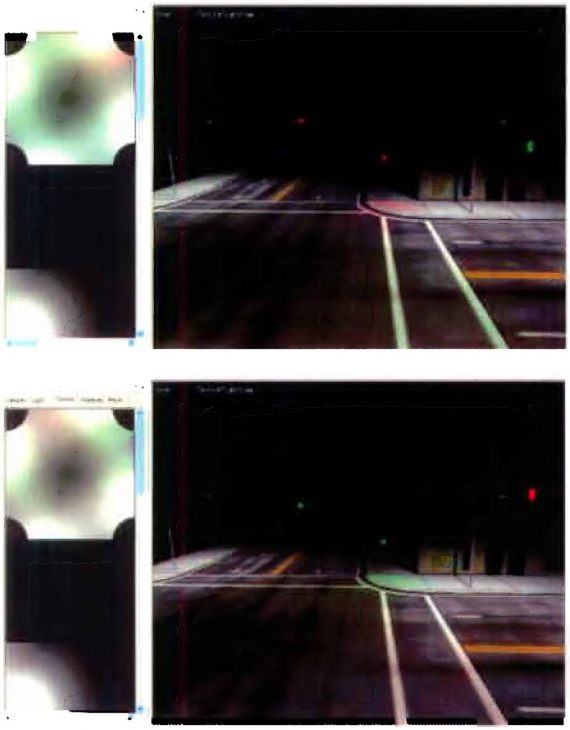

Figure 18 - Intersection Light Map State

Figure 18 shows an example of a traffic semaphore casting a colored haze in an intersection based on its state. The image in the upper left of each figure shows the portion of the Light Map representing the intersection quad. At runtime the Scene Graph API can be used to switch between the collection of Light Maps for this tile.

A logical extension of this technique is to render each tile (or database) multiple times with a varying "time of day." This would allow a user to specify the desired time of day, causing the system to load the appropriate set of texture maps to match the selection.

Specular Highlights and Environment Mapping

Given that specular lighting effects are directional in nature they can not be realistically pre-rendered into Light Maps. Depending on the desired effect it is possible to use a third texture unit and a Cube Map to achieve certain view-dependent effects. Figure 19 shows an example of an exaggerated specular highlight in a light mapped database.

Figure 19 - Specular Highlight

This database was rendered with a single directional light positioned high in the north-eastern sky. Light maps were rendered and applied, showing shadows being cast from the buildings to the road at the appropriate angle. A cube map was also constructed placing the "specular component" of the directional light source on the appropriate cube map faces. The cube map could also have contained an environment map if that was the desired effect. Figure 20 shows the cube map that was used. The sphere shows how the cube map would project into the scene, and the lower images show the cube map faces.

Figure 20 - Cube Map Specular

The cube map is mapped to the geometry using the GL_REFLECTION_MAP TexGen mode, and GL_ADD TexEnv mode. If using the fixed function pipeline it will likely be necessary to re-order the texture stages to satisfy the OpenGL lighting equation in Figure 5. It specifies that the sum of the specular and diffuse lighting terms should be multiplied by the decal texture, rather than the specular term being added to the product of the diffuse and decal texture. IE:

Right: `(diffuse + specular) * decal`

Wrong: `decal * diffuse + specular`

Caveats

When using this technique there are several things to consider. Specular highlights will generally not appear within shadows, and local light sources do not remain at a fixed relative angle to the view vector. Infinite directional lights, such as the Sun, work well because by definition they do stay at a fixed relative angular position to the viewing vector.

Half Life 2 pre-renders a cube map containing specular highlights and environment for each "room" within a level, from a fixed position. The specular highlights are technically only correct from that fixed position, but this is not usually evident during game play. This same technique could be used in a VisSim database to provide reasonable directional lighting effects.

Other Dynamic Effects

RTI is currently investigating several other dynamic techniques for use with Light Mapping including Virtualized OpenGL Light Sources, Headlight Effects and Texture Sampling to determine when an entity is "in-shadow." These efforts are ongoing, and will not be presented here.

Static Effects

Soft Coronas

Coronas are the fuzzy auras that appear around bright light sources, especially at night. They occur due to light refraction from particles in the air or from viewing the light through glass. In addition to the natural expectation that coronas will be present, they also provide some locality to light sources in that they help the user position the light source in space.

While there is a mathematical relationship between the size of the corona, the density of any water particles around the light source, and which "ring" the particle is in (using certain approximations), simulating coronas mathematically would be expensive. Instead, a suitable soft corona image texture can be used. Figure 21 shows one such corona image.

Figure 21 - Corona Texture

A plethora of corona texture maps can be found on the internet.

Soft coronas can be implemented using OpenGL's "Point Sprite" and "Point Parameters" extensions. These extensions allow OpenGL points to be textured and size attenuated based on distance from the viewer. Alternatively, the corona geometry could be added as a billboarded quad.

As the OpenFlight file is loaded, coronas are placed at the origin of each light source found in the database. The vertex colors of the corona are set to match the light source diffuse color, and its size is attenuated using the OpenFlight attenuation parameters.

Scenes with and without coronas are shown in Figure 22.

Figure 22 – With/Without Corona Effects

The scene contains six light sources, but this is not evident from the upper image due to the fact that the terrain below the light sources is not visible from this distance. In these scenes the corona geometry is drawn after all other geometry to make it appear to "shine through" the environment, as it would in real life. Randomness to the size of the geometry, as well as small variations in the color of the light sources can add additional realism to the scene.

Conclusion

Light Mapping is an efficient lighting technique that can be readily applied to OpenFlight databases and today's Image Generation hardware. Several popular Open Source Scene Graphs, as well as several proprietary ones, support the multi-texturing effects required to achieve Light Mapping with OpenFlight files. Offline tools are available to allow databases to be pre-rendered to include diffuse lighting effects from potentially thousands of light sources. Runtime cost of the technique remains constant regardless of the number of light sources. Databases may be rendered multiple times to provide "time of day" and other light state effects.